Science Publications

Universal Journal of Computer Sciences and Communications, 2022, 1, 531
www.scipublications.org/journal/index.phpo/ujcsc
DOI: 10.31586/ujcsc.2022.531

Review Article

A Problem of Accuracy of Computer Calculations

Rostyslav V. Bilous [1], Ivan H. Krykun [1,2,*]

[1] Department of Applied Mathematics, Vasyl' Stus Donetsk National University, Vinnytsia, Ukraine

[2] Department of Theory of Control Systems of Institute of Applied Mathematics and Mechanics of NAS of Ukraine, Sloviansk, Ukraine

*Correspondence: Ivan H. Krykun (iwanko@i.ua)

Abstract: The paper presented the results of the research related to the analysis of the reliability of computer calculations. Relevant examples of incorrect program operation were demonstrated: both quite simple and much less obvious, such as S. Rump's example. In addition to mathematical explanations, authors focused on purely software capabilities for controlling the accuracy of complex calculations. For this purpose, examples of effective use of the functionality of the decimal and fraction modules in Python 3.x were given.

Keywords: Computer Calculations, Errors, Programming, Floating Point Numbers

1. Introduction

The problem of reliability (accuracy) of computer calculations is one of the fundamental problems of computer science [1, 2]. Physical limitations on the amount of memory allocated by computer technology for number processing impose fundamental limitations on the possibilities and logic of organizing computer calculations, which also requires the involvement of specific mathematical research methods.

At the same time, the problem is not only the possibility of an incorrect result of theoretical research. As practice shows, the consequences of incorrect computer calculations can be catastrophic. There is a famous case: on February 25, 1991, during the Gulf War I (Operation Desert Storm), a "Patriot" air defence system failed to intercept an enemy missile, and the projectile hit a barracks of US soldiers, killing 28 people. An official investigation [3] showed that 24-bit interceptor processors make a time conversion error of 0,013 seconds every hour. "Patriot" was not restarted for more than 100 hours, and as a result of the accumulation of such seemingly minor errors, during this time there was an error in calculating the position of the missile at 600 meters.

Almost any academic textbook about calculation methods – an integral discipline of the training course for both physical and mathematical specialists and computer science specialists – usually contains only mathematical information about errors and the accuracy limits of calculations. The applied side of the issue, i.e. the computer implementation of calculations, remains out of consideration. Technical disciplines, such as "Programming", usually provide only the basic concepts of converting numbers to a binary code and back, and provide only general information about the existing standards for representing floating-point numbers.

Ignoring the fundamentals of binary arithmetic and the principles of the IEEE-754 standard [4] can significantly affect the correctness of both theoretical research (if software or computer equipment was used for modelling) and simply the quality of the product when developing application programs.

Programs are implemented in C++, Java, or Python programming languages, depending on when it is more appropriate: in C++ and Java, we can visually experiment by

How to cite this paper:
V. Bilous, R., & H. Krykun, I. (2022). A Problem of Accuracy of Computer Calculations. *Universal Journal of Computer Sciences and Communications*, *1*(1), 35–40. Retrieved from https://www.scipublications.com/journal/index.php/ujcsc/article/view/531

Received: xx xx, 2022
Accepted: xx xx, 2022
Published: xx xx, 2022

Copyright: © 2022 by the authors. Submitted for possible open access publication under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).

changing the types of floating-point numbers, float and double; in Python, by default, "long" arithmetic is implemented, that is, a set of algorithms for bitwise work with numbers of arbitrary length.

2. Examples of errors in computer calculations

Consider the following example of a feature of computer calculations. Let's check with the help of the C++ programming language, whether the distributive law is fulfilled in computer calculations. Two simple program codes for this can be seen in Figure 1 below

```
#include <iostream>
using namespace std;
int main() {
    double a = 0.987;
    double b = 0.278;
    double c = 1.268;
    double expr_1 = a*c + b*c;
    double expr_2 = (a + b)*c;
    cout<<(expr_1 == expr_2);
}
```

(**a**)

```
#include <iostream>
using namespace std;
int main() {
    float a = 0.9871;
    float b = 0.2781;
    float c = 1.268;
    float expr_1 = a*c + b*c;
    float expr_2 = (a + b)*c;
    cout<<(expr_1 == expr_2);
}
```

(**b**)

Figure 1. Program codes in the C++ programming language for checking of the fulfilment the distributive law: **(a)** for numbers 0,987, 0,278, and 1,268 in *double* format; **(b)** for numbers 0,9871, 0,2781, and 1,268 in *float* format

As expected, for code in Figure 1 (a) we get a result of **1** (i.e. $true$).

Let's slightly change the values of the numbers a and b, increasing them by 0,0001 (that is, we will change the numbers a and b by 0,01% and 0,04%, respectively). As a result, we will get a result of **0** ($false$).

If we output to the console the corresponding values for the case $a = 0{,}9871$ and $b = 0{,}2781$ with an accuracy of 20 decimal places, the program will issue

$$a \cdot c + b \cdot c \to 1{,}60427359999999996631,$$

$$(a + b) \cdot c \to 1{,}60427360000000018836.$$

So we get different values, which are insignificant but differ from the real value of 1,6042736 (the absolute error is $3{,}4 \cdot 10^{-17}$ and $1{,}9 \cdot 10^{-16}$).

Let's change the types of variables in the program to *float* (Figure 1 (b)), and as a result, we will already get the result $true$. That is, it is the reduction in the accuracy of the calculations that returns us to the correct result from a mathematical point of view, which at first glance looks somewhat paradoxical.

3. Representation of numbers in the IEEE-754 standard

The above complexities of computer calculations arise from the way real numbers are represented in computer memory, regulated by IEEE-754 standards.

The description of the IEEE-754 standard goes beyond the scope of this work, so we will limit ourselves to the demonstration of the representation of numbers from the example described above in *float* and *double* formats.

Float format: 32 bits are allocated to the number: the first bit is the sign bit (if the sign bit is 0, the number is non-negative (positive or zero); if the sign bit is 1 then the number is negative), the next 8 bits are allocated to the exponent, and the last 23 bits are allocated to the normalized mantissa. This is shown schematically in Figure 2 below.

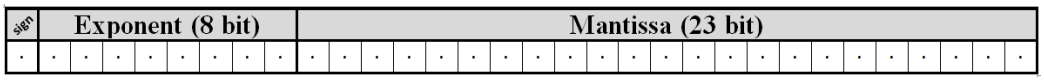

Figure 2. Representation of a number in *float* format in the IEEE-754 standard

Double format (i.e. double-precision format): 64 bits are allocated to the number: the first bit is the sign bit – the bit that indicates the sign of a number; 11 bits are allocated to the exponent, and the last 52 bits are accordingly allocated to the normalized mantissa (Figure 3).

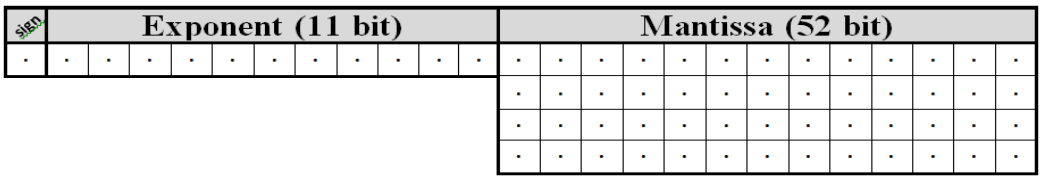

Figure 3. Representation of a number in the *double* format in the IEEE-754 standard

A simple example of comparing these two formats is comparing the same number in different formats using the Java 8 programming language.

```
public class MyClass {
    public static void main(String args[]) {
      float a = 0.3f;
      double b = 0.3;
      System.out.println("a == b >> " + (a == b));
      System.out.println("a-b >> " + (a-b));
    }
}
```

Figure 4. The program for comparing of the number 0,3 presented in different formats

This example compares the number 0,3 represented in *float* and *double* formats.

```
a == b >> false
a-b >> 1.1920928966180355E-8
```

Figure 5. The result of the program from Figure 4

As a result, we can see that the absolute values of two seemingly identical numbers are not equal to each other and have a significant difference when it comes to the accuracy of calculations.

4. The Rump's example

Next, we will consider the example proposed by S. Rump [5] back in 1988. It is necessary to calculate the value of the expression

$$f(x,y) = 333{,}75 \cdot y^6 + x^2 \cdot (11 \cdot x^2y^2 - y^6 - 121 \cdot y^4 - 2) + 5{,}5 \cdot y^8 + \frac{x}{2y}$$

at the point $(77617\,;\ 33096)$.

The task looks quite simple. If we temporarily ignore the last term $\frac{x}{2y}$ (which is not really important), and considering that $y = 33096$ is an even number, it is obvious that the value of the expression $f(x,y) - \frac{x}{2y}$ in general must be an integer.

To implement the Rump's example, we will use Python 3

```
x=77617
y=33096
print (333.75*y**6+x**2*(11*x**2-y**2-y**6-121*y**4-2)+5.5*y**8+0.5*x/y)
```

Figure 6. The program implementation of the Rump's example

The result of executing the program in Figure 6 will be the value 11805916207174113034 24, although in fact

$$f(77617,33096) = -\frac{54767}{66192}$$

The characteristic problem is precisely at the point $(77617;\ 33096)$.

Let's explain what is special about this point. Let's separate the positive and negative parts of the expression (as already mentioned: the last addition of $\frac{x}{2y}$ does not really significantly affect the correctness of the calculation result). Let's separate the positive and negative parts of the expression $(x,y) - \frac{x}{2y}$, as shown in Figure 7.

```
x=77617
y=33096
positive = 1335*y**6//4+x**2*11*x**2*y**2+11*y**8//2
negative = x**2*y**6+121*x**2*y**4+2*x**2
print(positive, negative, sep="\n")
```

Figure 7. The program implementation of the Rump's example with separated positive and negative parts

We will get the result

$$positive = 79171122165662886468976131642684998 4;$$

$$negative = 79171122165662886468976131642684998 6.$$

That is, in our example

$$positive - negative = -2,$$

so final result coincides with the mathematical one. The fact is that one of the main problems when working with floating-point numbers is the subtraction operation, because when subtracting numbers close in value, significant digits may be lost, and the result in this case will be incorrect.

If we change one line in the program by replacing integer division (//) with regular division (/), that is

$$positive = \frac{1335 \cdot y^6}{4} + x^2 \cdot (11 \cdot x^2 y^2) + \frac{11 \cdot y^8}{2}$$

so the value of the variable *positive* will be equal to

$$positive = 79171122165662886297210756321299660 80$$

and the difference between the values of *positive* and *negative* will be already

$$-34968685684296883906\,.$$

If we rewrite the expression for the variable *positive* in a format more familiar to the Python user

$$positive = \frac{1335}{4}\cdot y^6 + x^2\cdot(11\cdot x^2\cdot y^2) + \frac{11}{2}\cdot y^8,$$

then the value of *positive* will be equal to

$$positive \;=\; 79171122165662898103126963495412695 04$$

since different rounding errors are given in signs for different arithmetic operations (as we already noted in the previous section: the order of operations also matters for the accuracy of calculations). The difference in this case between the received positive value and the previous one will be 1180591620717411303424, and the value

$$positive - negative \;=\; 1145622935033114419518\,.$$

Correctly change the sequence of performing arithmetic operations and immediately get errors of the order 10^{21}.

5. Conclusions and prospects for further research

It should be noted that Python has the ability to a certain extent to eliminate some problems arising from calculations with floating-point numbers – for example, using the functionality of the decimal module to control the accuracy of calculations. The official documentation for the decimal module can be found at [6]. Without going into unnecessary details, we note that the decimal module (in particular, the Decimal class) makes it possible to work with floating-point numbers as with ordinary fractions.

With all of the above, for example, only using the capabilities of the Decimal class seems insufficient. Skipping several intermediate stages of the research, the author found a way to get the correct result of software calculations. This method consists in the joint use of the decimal module together with the application of the multiplication reduction scheme (Horner scheme), with the aim of reducing the number of arithmetic operations and increasing the accuracy of calculations. In this case, we have to rewrite the expression $f(x,y) - \frac{x}{2y}$ in the form

$$f(x,y) = y^2\cdot\left(y^2\cdot\left(y^2\cdot\left(5{,}5\cdot y^2(333{,}75 - x^2)\right) - 121\cdot x^2\right) + 11\cdot x^4\right) - 2\cdot x^2.$$

We implement the described approach programmatically (Figure 8).

```
x=77617
y=33096
x2=Decimal(x)**2
y2=Decimal(y)**2
y4=y2**2
x4=x2**2
print(y2*(y2*(y2*(Decimal(5.5)*y2+(Decimal(333.75)-x2))-121*x2)+11*x4)-2*x2)
```

Figure 8. The program implementation of the example using the reduced multiplication scheme

In the result we get expected value -2.

The described examples can be called linear, since the reasons for the occurrence of calculation features become obvious even with a cursory analysis, if we take into account the IEEE-754 standard.

The question of using the results mentioned in the paper, in particular, in the modelling of random and natural processes was considered in recent studies of such processes [7-16].

Author Contributions: All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: The authors express their heartfelt gratitude to the brave soldiers of the Ukrainian Armed Forces who protect the lives of the authors and their families from Russian bloody murderers since 2014.

Conflicts of Interest: The authors declare no conflict of interest.

References

[1] Kopec, D. *Classic Computer Science in Phyton*, Manning Publications: Shelter Island, NY, USA, 2019.
[2] Kopec, D. *Classic Computer Science in Java*, Manning Publications: Shelter Island, NY, USA, 2020.
[3] Patriot Missile Defense: Software Problem Led to System Failure at Dhahran, Saudi Arabia: IMTEC-92-26, 1992. Available online: https://www.gao.gov/assets/220/215614.pdf (accessed on 25/09/2022).
[4] IEEE Standard for Floating-Point Arithmetic. Available online: https://irem.univ-reunion.fr/IMG/pdf/ieee-754-2008.pdf (accessed on 25/09/2022).
[5] Rump, S.M. Algorithms for Verified Inclusions: Theory and Practice. In: *Reliability in Computing: The Role of Interval Methods in Scientific Computing*; Moore, R. E. (ed.) Academic Press: Boston, USA, 1988; pp. 109–126.
[6] Decimal fixed point and floating point arithmetic. Available online: https://docs.python.org/3/library/decimal.html (accessed on 25/09/2022).
[7] Krykun, I.H. Large deviation principle for stochastic equations with local time. *Theory of Stochastic Processes* **2009**. *15(31), No. 2.* pp. 140–155.
[8] Krykun, I.H. Functional law of the iterated logarithm type for a skew Brownian motion. *Teorija Imovirnostej ta Matematychna Statystyka* **2012**, *87*. pp. 60-77 (*in Ukrainian*)
[9] Krykun, I.H.; Makhno, S.Ya. The Peano phenomenon for Itô equations. *Journal of Mathematical Sciences* **2013**, *192, Issue 4*, pp. 441–458. DOI: 10.1007/s10958-013-1407-5
[10] Krykun, I.H. Functional law of the iterated logarithm type for a skew Brownian motion. *Theory of Probability and Mathematical Statistics* **2013**, *87*, pp. 79–98. DOI: 10.1090/S0094-9000-2014-00906-0
[11] Krykun, I.H. Convergence of skew Brownian motions with local times at several points that are contracted into a single one. *Journal of Mathematical Sciences* **2017**, *221, Issue 5*, pp. 671–678. DOI: 10.1007/s10958-017-3258-y
[12] Krykun, I.H. The Arc-Sine Laws for the Skew Brownian Motion and Their Interpretation. *Journal of Applied Mathematics and Physics* **2018**, *6, No. 2*, pp. 347–357. DOI: 10.4236/jamp.2018.62033
[13] Krykun, I. H. The Arctangent Regression and the Estimation of Parameters of the Cauchy Distribution. *Journal of Mathematical Sciences* **2020**, *249, Issue 5*, pp. 739-753.
[14] Krykun, I.H. The arcsine laws in the modelling of the natural processes depending on random factors. In *Physical and mathematical justification of scientific achievements: collective monograph*, Primedia eLaunch LLC: Boston, USA, **2020**; pp. 24-33. DOI: 10.46299/ISG.2020.MONO.PHYSICAL.III
[15] Krykun, I.H. New Approach to Statistical Analysis of Election Results. *International Journal of Mathematical, Engineering, Biological and Applied Computing* **2022**, *1, No. 2*, pp. 68–76. DOI: 10.31586/ijmebac.2022.466
[16] Cherniichuk, H.P.; Krykun, I.H. Notes about Winning Strategies for Some Combinatorial Games. *Journal of Mathematics Letters* **2022**, *1 No.1*, pp.1–9. DOI: 10.31586/jml.2022.496